Intel® Celeron® Processor in the 478-Pin Package Specification Update

Release Date: September 2002

Order Number: 290749-004

The Intel® Celeron® processor in the 478-pin package may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

#

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
May 2002	-001	Initial release.
June 2002	-002	Added erratum V35. Added Documentation Changes V4-V5. Updated processor identification information table.
July 2002	-003	Added erratum V37. Added Documentation Changes V3-V12.
September 2002	-004	Updated with Intel ® Celeron ® Processor on .13 Micron Process and in the 478-Pin Package. Added erratum V38. Updated Erratum V17. Added Documentation Changes V3-V24.

PREFACE

This document is an update to the specifications contained in the following document:

- *Intel® Celeron® Processor in the 478-Pin Package Datasheet*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3 (Order Numbers 245470, 245471, and 245472, respectively)*

It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs and Errata.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the Intel® Celeron® processor in the 478-pin package's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

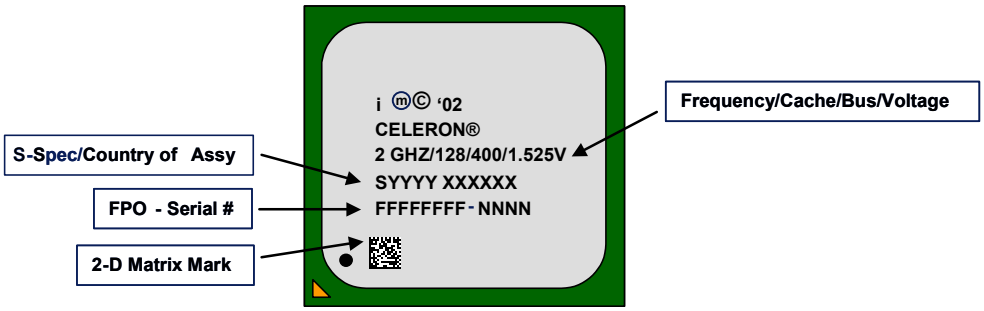
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Specification Changes are modifications to the current published specifications for the Intel Pentium 4 processor. These changes will be incorporated in the next release of the specifications.

Specification Update for the Intel® Celeron® Processor in the 478-Pin Package

GENERAL INFORMATION

Intel® Celeron® Processor on .13 Micron Process and/or in the 478-Pin Package



IDENTIFICATION INFORMATION

The Intel® Celeron® processor in the 478-pin package in can be identified by the following values:

Family[1]	Model[2]	Brand ID[3]
1111	0001	00001010
1111	0010	00001010

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. The Brand ID corresponds to bits [7:0] of the EBX register after the CPUID instruction is executed with a 1 in the EAX register.

S-Spec	Core Stepping	L2 Cache Size (bytes)	PROCESSOR SIGNATURE	Speed Core/Bus	Package and Revision	Notes
SL69Z	E0	128K	0F13h	1.70GHz/400MHz	FC-PGA2 35.0 mm, Rev 02	1,2
SL68C	E0	128K	0F13h	1.70GHz/400MHz	FC-PGA2 35.0 mm, Rev 02	1
SL6A2	E0	128K	0F13h	1.80GHz/400MHz	FC-PGA2 35.0 mm, Rev 02	2,3
SL68D	E0	128K	0F13h	1.80GHz/400MHz	FC-PGA2 35.0 mm, Rev 02	3
SL6LC	C1	128K	0F27h	2 GHz/400MHz	FC-PGA2 31.0 mm , rev 1.0	2,4
SL6HY	C1	128K	0F27h	2 GHz/400MHz	FC-PGA2 31.0 mm , rev 1.0	4

NOTES:

1. This processor has maximum Tcase of 76 deg C.
2. This is a boxed processor with an unattached fan heatsink.
3. This processor has maximum Tcase of 77 deg C.
4. This processor has maximum Tcase of 68 deg C

SUMMARY OF CHANGES

The following table indicates the Errata that apply to Intel® Celeron® processor in the 478-pin package. Intel intends to fix some of the errata in a future stepping of the component. This table uses the following notations:

CODES USED IN SUMMARY TABLE	
X	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
PlanFix:	This erratum may be fixed in a future stepping of the product.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Document change or update that will be implemented.
PKG:	This column refers to errata on the Intel® Celeron® processor in the 478-pin package substrate.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item is prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Mobile Intel® Pentium® II processor

C = Intel® Celeron® processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

G = Intel® Pentium® III Xeon™ processor

H = Mobile Intel® Celeron® processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Mobile Intel® Pentium® III Processor - M

M = Mobile Intel® Celeron® processor

N = Intel® Pentium® 4 processor

O = Intel® Xeon™ processor MP

P = Intel® Xeon™ processor

T = Mobile Intel® Pentium® 4 processor – M

V = Intel® Celeron® processor in the 478-Pin Package

Summary of Errata

NO.	E0	NC1	Plans	ERRATA
V1	X	X	NoFix	I/O restart in SMM may fail after simultaneous machine check exception (MCE)
V2	X	X	NoFix	MCA registers may contain invalid information if RESET# occurs and PWRGOOD is not held asserted
V3	X		Fixed	Uncacheable (UC) code in same line as write back (WB) data may lead to data corruption
V4	X	X	NoFix	Transaction is not retried after BINIT#
V5	X	X	NoFix	Invalid opcode 0FFFh requires a ModRM byte
V6	X	X	NoFix	FSW may not be completely restored after page fault on FRSTOR or FLDENV instructions
V7	X	X	NoFix	The processor flags #PF instead of #AC on an unlocked CMPXCHG8B instruction
V8	X	X	NoFix	When in no-fill mode the memory type of large pages are incorrectly forced to uncacheable
V9	X	X	NoFix	Processor may hang due to speculative page walks to non-existent system memory
V10	X		Fixed	Writing a performance counter may result in incorrect value
V11	X	X	NoFix	IA32_MC0_STATUS register overflow bit not set correctly
V12	X		Fixed	Performance counter may contain incorrect value after being stopped
V13	X		NoFix	MCA error code field in IA32_MC0_STATUS register may become out of sync with the rest of the register
V14	X		NoFix	The IA32_MC1_STATUS register may contain incorrect information for correctable errors
V15	X	X	NoFix	Debug mechanisms may not function as expected
V16	X	X	NoFix	Machine check architecture error reporting and recovery may not work as expected
V17	X		Fixed	Processor may timeout waiting for a device to respond after ~0.67 seconds
V18	X	X	NoFix	Cascading of performance counters does not work correctly when forced overflow is enabled
V19	X		Fixed	IA32_MC1_STATUS MSR ADDRESS VALID bit may be set when no valid address is available
V20	X	X	NoFix	EMON event counting of x87 loads may not work as expected
V21	X		Fixed	Software controlled clock modulation using a 12.5% or 25% duty cycle may cause the processor to hang
V22	X		Fixed	SQRTPD and SQRTSD may return QnaN indefinite instead of negative zero
V23	X	X	PlanFix	Bus Invalidate Line requests that return unexpected data may result in L1 cache corruption

Summary of Errata

NO.	E0	NC1	Plans	ERRATA
V24	X	X	PlanFix	Write Combining (WC) load may result in unintended address on system bus
V25	X		Fixed	Incorrect data may be returned when page tables are in Write Combining (WC) memory space
V26	X		PlanFix	Buffer on resistance may exceed specification
V27	X	X	NoFix	Processor issues inconsistent transaction size attributes for locked operation
V28	X		Fixed	Multiple accesses to the same S-state L2 cache line and ECC error combination may result in loss of cache coherency
V29	X		Fixed	Processor may hang when resuming from Deep Sleep state
V30	X	X	NoFix	When the processor is in the System Management Mode (SMM), debug registers may be fully writeable
V31	X	X	NoFix	Associated counting logic must be configured when using Event Selection Control (ESCR) MSR
V32	X	X	NoFix	IA32_MC0_ADDR and IA32_MC0_MISC registers will contain invalid or stale data following a Data, Address, or Response Parity Error
V33	X		Fixed	CR2 may be incorrect or an incorrect page fault error code may be pushed onto stack after execution of an LSS instruction
V34	X	X	NoFix	System may hang if a fatal cache error causes Bus Write Line (BWL) transaction to occur to the same cache line address as an outstanding Bus Read Line (BRL) or Bus Read-Invalidate Line (BRIL)
V35	X		Fixed	Processor Does not Flag #GP on Non-zero Write to Certain MSRs
V36	X		Fixed	L2 cache may contain stale data in the Exclusive state
V37	X	X	NoFix	Simultaneous assertion of A20M# and INIT# may result in incorrect data fetch
V38	X	X	No Fix	Glitches on Address and Data Strobe Signals May Cause System Shutdown

NO.	E0	Plans	DOCUMENTATION CHANGES
V1	X	Doc	SSE and SSE2 Instructions Opcodes
V2	X	Doc	Executing the SSE2 Variant on a Non-SSE2 Capable Processor
V3	X	Doc	Direction Flag (DF) Mistakenly Denoted as a System Flag
V4	X	Doc	Fopcode Compatibility Mode
V5	X	Doc	FCOS, FPTAN, FSIN, and FSINCOS Trigonometric Domain not correct
V6	X	Doc	Incorrect Description of stack
V7	X	Doc	EFLAGS Register Correction
V8	X	Doc	PSE-36 Paging Mechanism
V9	X	Doc	0x33 Opcode
V10	X	Doc	Incorrect Information for SLDT
V11	X	Doc	LGDT/LIDT Instruction Information Correction
V12	X	Doc	Errors In Instruction Set Reference
V13	X	Doc	RSM Instruction Set Summary
V14	X	Doc	Correct MOVAPS and MOVAPD Operand Section
V15	X	Doc	DAA—Decimal Adjust AL after Addition
V16	X	Doc	DAS—Decimal Adjust AL after Subtraction
V17	X	Doc	Omission of Dependency between BTM and LBR
V18	X	Doc	I/O Permissions Bitmap Base Addy > 0xDFFF Does not Cause #GP(0) Fault
V19	X	Doc	Wrong Field Width for MINSS and MAXSS
V20	X	Doc	Figure 15-12 PEBS Record Format
V21	X	Doc	I/O Permission Bit Map
V22	X	Doc	Cache Description
V23	X	Doc	Instruction Formats and Encoding
V24	X	Doc	Machine-Check Initialization

NO.	E0	Plans	SPECIFICATION CLARIFICATIONS
			No Update

NO.	E0	Plans	SPECIFICATION CHANGES
			No Update

ERRATA

V1. I/O Restart in SMM may Fail after Simultaneous Machine Check Exception (MCE)

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the processor will signal a Machine Check Exception (MCE). If the instruction is directed at a device that is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have an incorrect state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and board I/O restart, the first thing the SMM handler code should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the MCE handler to execute. If there is no MCE pending, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V2. MCA Registers may Contain Invalid Information if RESET# Occurs and PWRGOOD is not Held Asserted

Problem: This erratum can occur as a result either of the following events:

- PWRGOOD is de-asserted during a RESET# assertion causing internal glitches that may result in the possibility that the MCA registers latch invalid information.
- Or during a reset sequence if the processor's power remains valid regardless of the state of PWRGOOD, and RESET# is re-asserted before the processor has cleared the MCA registers, the processor will begin the reset process again but may not clear these registers.

Implication: When this erratum occurs, the information in the MCA registers may not be reliable.

Workaround: Ensure that PWRGOOD remains asserted throughout any RESET# assertion and that RESET# is not re-asserted while PWRGOOD is de-asserted.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V3. Uncacheable (UC) Code in Same Line as Write Back (WB) Data may Lead to Data Corruption

Problem: When both code (being accessed as UC or WC) and data (being accessed as WB) are aliased into the same cache line, the UC fetch will cause the processor to self-snoop and generate an implicit writeback. The data supplied by this implicit writeback may be corrupted due to the way the processor handles self-modifying code.

Implication: UC or WC code located in the same cache line as WB data may lead to data corruption.

Workaround: UC or WC code should not be located in the same physical 64 byte cache line as any location that is being stored to with WB data.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V4. Transaction is not Retried after BINIT#

Problem: If the first transaction of a locked sequence receives a HITM# and DEFER# during the snoop phase it should be retried and the locked sequence restarted. However, if BINIT# is also asserted during this transaction, it will not be retried.

Implication: When this erratum occurs, locked transactions will unexpectedly not be retried.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V5. Invalid Opcode 0FFFh Requires a ModRM Byte

Problem: Some invalid opcodes require a ModRM byte (or other following bytes), while others do not. The invalid opcode 0FFFh did not require a ModRM byte in previous generation Intel architecture processors, but does in the Intel® Pentium® 4 processor.

Implication: The use of an invalid opcode 0FFFh without the ModRM byte may result in a page or limit fault on the Intel® Pentium® 4 processor.

Workaround: Use a ModRM byte with invalid 0FFFh opcode.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V6. FSW may not be Completely Restored after Page Fault on FRSTOR or FLDENV Instructions

Problem: If the FPU operating environment or FPU state (operating environment and register stack) being loaded by an FLDENV or FRSTOR instruction wraps around a 64Kbyte or 4Gbyte boundary and a page fault (#PF) or segment limit fault (#GP or #SS) occurs on the instruction near the wrap boundary, the upper byte of the FPU status word (FSW) might not be restored. If the fault handler does not restart program execution at the faulting instruction, stale data may exist in the FSW.

Implication: When this erratum occurs, stale data will exist in the FSW.

Workaround: Ensure that the FPU operating environment and FPU state do not cross 64Kbyte or 4Gbyte boundaries. Alternately, ensure that the page fault handler restarts program execution at the faulting instruction after correcting the paging problem.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V7. The Processor Flags #PF Instead of #AC on an Unlocked CMPXCHG8B Instruction

Problem: If a data page fault (#PF) and alignment check fault (#AC) both occur for an unlocked CMPXCHG8B instruction, then #PF will be flagged.

Implication: Software that depends #AC before #PF will be affected since #PF is flagged in this case.

Workaround: Remove the software's dependency on the fact that #AC has precedence over #PF. Alternately, reload the page in the page fault handler and then restart the faulting instruction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V8. When in No-Fill Mode the Memory Type of Large Pages are Incorrectly Forced to Uncacheable

Problem: When the processor is operating in No-Fill Mode (CR0.CD=1), the paging hardware incorrectly forces the memory type of large (PSE-4M and PAE-2M) pages to uncacheable (UC) memory type regardless of the MTRR settings. By forcing the memory type of these pages to UC, load operations, which should hit valid data in the L1 cache, are forced to load the data from system memory. Some applications will lose the performance advantage associated with the caching permitted by other memory types.

Implication: This erratum may result in some performance degradation when using no-fill mode with large pages.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V9. Processor may Hang due to Speculative Page Walks to Non-Existent System Memory

Problem: A load operation that misses the Data Translation Lookaside Buffer (DTLB) will result in a page-walk. If the page-walk loads the Page Directory Entry (PDE) from cacheable memory and that PDE load returns data that points to a valid Page Table Entry (PTE) in uncacheable memory the processor will access the address referenced by the PTE. If the address referenced does not exist the processor will hang with no response from system memory.

Implication: Processor may hang due to speculative page walks to non-existent system memory.

Workaround: Page directories and page tables in UC memory space which are marked valid must point to physical addresses that will return a data response to the processor.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V10. Writing a Performance Counter may Result in Incorrect Value

Problem: When a performance counter is written and the event counter for the event being monitored is non-zero, the performance counter will be incremented by the value on that event counter. Because the upper eight bits of the performance counter are not written at the same time as the lower 32 bits, the increment due to the non-zero event counter may cause a carry to the upper bits such that the performance counter contains a value about four billion (2^{32}) higher than what was written.

Implication: When this erratum occurs, the performance counter will contain a different value from that which was written.

Workaround: If the performance counter is set to select a null event and the counter configuration and control register (CCCR) for that counter has its compare bit set to zero, before the performance counter is written, this erratum will not occur. Since the lower 32 bits will always be correct, event counting which does not exceed 2^{32} events will not be affected.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V11. IA32_MC0_STATUS Register Overflow Bit not Set Correctly

Problem: The Overflow Error bit (bit 62) in the IA32_MC0_STATUS register indicates, when set, that a machine check error occurred while the results of a previous error were still in the error reporting bank (i.e. the valid bit was set when the new error occurred). In the case of this erratum, if an uncorrectable error is logged in the error-reporting bank and another error occurs, the overflow bit will not be set.

Implication: When this erratum occurs the overflow bit will not be set.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V12. Performance Counter may Contain Incorrect Value after being Stopped

Problem: If a performance counter is stopped on the precise internal clock cycle where the intermediate carry from the lower 32 bits of the counter to the upper eight bits occurs, the intermediate carry is lost.

Implication: When this erratum occurs, the performance counter will contain a value about 4 billion (2^{32}) less than it should.

Workaround: Since this erratum does not occur if the performance counters are read when running, a possible workaround is to read the counter before stopping it. Since the lower 32 bits will always be correct, event counting which does not exceed 2^{32} events will not be affected.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V13. MCA Error Code Field in IA32_MC0_STATUS Register may become out of Sync with the Rest of the Register

Problem: The MCA Error Code field of the IA32_MC0_STATUS register gets written by a different mechanism than the rest of the register. For uncorrectable errors, the other fields in the IA32_MC0_STATUS register are only updated by the first error. Any subsequent errors cause the Overflow Error bit to be asserted until this register is cleared. Because of this erratum, any further errors that are detected will update the MCA Error Code field without updating the rest of the register, thereby leaving the IA32_MC0_STATUS register with stale information.

Implication: When this erratum occurs, the IA32_MC0_STATUS register contains stale information.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V14. The IA32_MC1_STATUS Register may Contain Incorrect Information for Correctable Errors

Problem: When a speculative load operation hits the L2 cache and receives a correctable error, the IA32_MC1_STATUS register may be updated with incorrect information. The IA32_MC1_STATUS register should not be updated for speculative loads.

Implication: When this erratum occurs, the IA32_MC1_STATUS register will contain incorrect information for correctable errors.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V15. Debug Mechanisms may not Function as Expected

Problem: Certain debug mechanisms may not function as expected on the processor. The cases are as follows:

- When the following conditions occur: 1) An FLD instruction signals a stack overflow or underflow, 2) the FLD instruction splits a page-boundary or a 64 byte cache line boundary, 3) the instruction matches a Debug Register on the high page or cache line respectively, and 4) the FLD has a stack fault and a memory fault on a split access, the processor will only signal the stack fault and the debug exception will not be taken.
- When a data breakpoint is set on the ninth and/or tenth byte(s) of a floating point store using the Extended Real data type, and an unmasked floating point exception occurs on the store, the break point will not be captured.
- When any instruction has multiple debug register matches, and any one of those debug registers is enabled in DR7, all of the matches should be reported in DR6 when the processor goes to the debug handler. This is not true during a REP instruction. As an example, during execution of a REP MOVSW instruction the first iteration a load matches DR0 and DR2 and sets DR6 as FFFF0FF5h. On a subsequent iteration of the instruction, a load matches only DR0. The DR6 register is expected to still contain FFFF0FF5h, but the processor will update DR6 to FFFF0FF1h.
- A data breakpoint that is set on a load to uncacheable memory may be ignored due to an internal segment register access conflict. In this case the system will continue to execute instructions, bypassing the intended breakpoint. Avoiding having instructions that access segment descriptor registers e.g. LGDT, LIDT close to the UC load, and avoiding serialized instructions before the UC load will reduce the occurrence of this erratum.

Implication: Certain debug mechanisms do not function as expected on the processor.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V16. Machine Check Architecture Error Reporting and Recovery may not Work as Expected

Problem: When the processor detects errors it should attempt to report and/or recover from the error. In the situations described below, the processor does not report and/or recover from the error(s) as intended.

- When a transaction is deferred during the snoop phase and subsequently receives a Hard Failure response, the transaction should be removed from the bus queue so that the processor may proceed. Instead, the transaction is not properly removed from the bus queue, the bus queue is blocked, and the processor will hang.
- When a hardware prefetch results in an uncorrectable tag error in the L2 cache, IA32_MC0_STATUS.UNCOR and IA32_MC0_STATUS.PCC are set but no Machine Check Exception (MCE) is signaled. No data loss or corruption occurs because the data being prefetched has not been used. If the data location with the uncorrectable tag error is subsequently accessed, an MCE will occur. However, upon this MCE, or any other subsequent MCE, the information for that error will not be logged because IA32_MC0_STATUS.UNCOR has already been set and the MCA status registers will not contain information about the error which caused the MCE assertion but instead will contain information about the prefetch error event.
- When the reporting of errors is disabled for Machine Check Architecture (MCA) Bank 2 by setting all IA32_MC2_CTL register bits to 0, uncorrectable errors should be logged in the IA32_MC2_STATUS register but no machine-check exception should be generated. Uncorrectable loads on bank 2, which would normally be logged in the IA32_MC2_STATUS register, are not logged.
- When one half of a 64 byte instruction fetch from the L2 cache has an uncorrectable error and the other 32 byte half of the same fetch from the L2 cache has a correctable error, the processor will attempt to correct the correctable error but cannot proceed due to the uncorrectable error. When this occurs the processor will hang.
- When an L1 cache parity error occurs, the cache controller logic should write the physical address of the data memory location that produced that error into the IA32_MC1_ADDR register. In some instances of a parity error on a load operation that hits the L1 cache, however, the cache controller logic may write the physical address from a subsequent load or store operation into the IA32_MC1_ADDR register.
- The local xAPIC has an Error Status Register which records all errors which it detects. Bit 6 of this register, the Receive Illegal Vector bit, is set when the local xAPIC detects an illegal vector in a message that it received. When an illegal vector error is received on the same internal clock that the error status register is being written due to a previous error, bit 6 does not get set and illegal vector errors are not flagged.
- If an instruction fetch results in an uncorrectable error and there is also a debug breakpoint at this address, the processor will livelock and the uncorrectable error will not be logged in the machine check registers.
- The MCA Overflow bit should be set when an uncorrectable error resides within the register bank (valid bit is already set) and any subsequent errors occur. The Overflow bit being set indicates that more than one error has occurred. Because of this erratum, if any further errors occur, the MCA Overflow bit will not be updated; thereby incorrectly indicating only one error has been received.

Implication: The processor is unable to correctly report and/or recover from certain errors.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V17. Processor may Timeout Waiting for a Device To Respond After 0.67 Seconds

Problem: The PCI 2.1 target initial latency specification allows two seconds for a device to respond during initialization-time. The processor may timeout after only approximately 0.67 seconds. When the processor times out it will hang with IERR# asserted. PCI devices that take longer than 0.67 seconds to initialize may not be initialized properly.

Implication: System may hang with IERR# asserted.

Workaround: Due to the long initialization time observed on some commercially available PCI cards, it may be necessary to disable the timeout counter during the PCI initialization sequence. This can be accomplished by temporarily setting Bit 5 of the MISC_ENABLES_MSR located at address 1A0H to 1. This model specific register (MSR) is software visible but should only be set for the duration of the PCI initialization sequence. It is necessary to re-enable the timeout counter by clearing this bit after completing the PCI initialization sequence. CAUTION: The processor's Thermal Monitor feature may not function if the timeout counter is not re-enabled after completing the PCI initialization.

After the system is fully initialized, this erratum may occur either when a PCI device is hot added into the system or when a PCI device is transitioned from D3 cold. System software responsible for completing the hot add and the power state transition from D3 cold should allow for a delay of the target initial latency prior to initiating configuration accesses to the PCI device.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section

V18. Cascading of Performance Counters does not work Correctly when Forced Overflow is Enabled

Problem: The performance counters are organized into pairs. When the CASCADE bit of the Counter Configuration Control Register (CCCR) is set, a counter that overflows will continue to count in the other counter of the pair. The FORCE_OVF bit forces the counters to overflow on every non-zero increment. When the FORCE_OVF bit is set, the counter overflow bit will be set but the counter no longer cascades.

Implication: The performance counters do not cascade when the FORCE_OVF bit is set.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V19. IA32_MC1_STATUS MSR ADDRESS VALID bit may be set when no Valid Address is Available

Problem: The processor should only log the address for L1 parity errors in the IA32_MC1_STATUS MSR if a valid address is available. If a valid address is not available, the ADDRESS VALID bit in the IA32_MC1_STATUS MSR should not be set. In instances where an L1 parity error occurs and the address is not available because the linear to physical address translation is not complete or an internal resource conflict has occurred, the ADDRESS VALID bit is incorrectly set.

Implication: The ADDRESS VALID bit is set even though the address is not valid.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V20. EMON Event Counting of x87 Loads may not Work as Expected

Problem: If a performance counter is set to count x87 loads and floating point exceptions are unmasked, the FPU Operand Data Pointer (FDP) may become corrupted.

Implication: When this erratum occurs, the FPU Operand Data Pointer (FDP) may become corrupted.

Workaround: This erratum will not occur with floating point exceptions masked. If floating point exceptions are unmasked, then performance counting of x87 loads should be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V21. Software Controlled Clock Modulation using a 12.5% or 25% Duty Cycle may cause the Processor to Hang

Problem: Processor clock modulation may be controlled via a processor register (IA32_THERM_CONTROL). The On-Demand Clock Modulation Duty Cycle is controlled by bits 3:1. If these bits are set to a duty cycle of 12.5% or 25%, the processor may hang while attempting to execute a floating-point instruction. In this failure, the last instruction pointer (LIP) is pointing to a floating-point instruction whose instruction bytes are in UC space and which takes an exception 16 (floating point error exception). The processor stalls trying to fetch the bytes of the faulting floating-point instruction and those following it. This processor hang is caused by interactions between thermal control circuit and floating-point event handler.

Implication: The processor will go into a sleep state from which it fails to return.

Workaround: Use a duty cycle other than 12.5% or 25%.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V22. SQRTPD and SQRTSD may Return QNaN Indefinite Instead of Negative Zero

Problem: When DAZ mode is enabled, and a SQRTPD or SQRTSD instruction has a negative denormal operand, the instruction will return a QNaN indefinite when the specified response should be a negative zero.

Implication: When this erratum occurs, the instruction will return a QNaN indefinite when a negative zero is expected.

Workaround: Ensure that negative denormals are not used as operands to the SQRTPD or SQRTSD instructions when DAZ mode is enabled. Software could enable FTZ mode to ensure that negative denormals are not generated by computation prior to execution of a SQRTPD or SQRTSD instruction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V23. Bus Invalidate Line Requests that Return Unexpected Data may Result in L1 Cache Corruption

Problem: When a Bus Invalidate Line (BIL) request receives unexpected data from a deferred reply, and a store operation write combines to the same address, there is a small window where the L0 is corrupt, and loads can retire with this corrupted data. This erratum occurs in the following scenario:

- A Read-For-Ownership (RFO) transaction is issued by the processor and hits a line in shared state in the L1 cache.
- The RFO is then issued on the system bus as a 0 length Read-Invalidate (a BIL), since it doesn't need data, just ownership of the cache line.
- This transaction is deferred by the chipset.
- At some later point, the chipset sends a deferred reply for this transaction with an implicit write-back response. For this erratum to occur, no snoop of this cache line can be issued between the BIL and the deferred reply.
- The processor issues a write-combining store to the same cache line while data is returning to the processor. This store straddles an 8-byte boundary.
- Due to an internal boundary condition, a time window exists where the L1 cache contains corrupt data which could be accessed by a load.

Implication: No known commercially available chipsets trigger the failure conditions.

Workaround: The chipset could issue a BIL (snoop) to the deferred processor to eliminate the failure conditions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V24. Write Combining (WC) Load May Result in Unintended Address on System Bus

Problem: When the processor performs a speculative write combining (WC) load, down the path of a mispredicted branch, and the address happens to match a valid UnCacheable (UC) address translation with the Data Translation Look-Aside Buffer, an unintended UnCacheable load operation may be sent out on the system bus.

Implication: When this erratum occurs, an unintended load may be sent on system bus. Intel has only encountered this erratum during pre-silicon simulation.

Workaround: It is possible for the BIOS to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V25. Incorrect Data May be Returned When Page Tables Are In Write Combining (WC) Memory Space

Problem: If page directories and/or page tables are located in Write Combining (WC) memory, speculative loads to cacheable memory may complete with incorrect data.

Implication: Cacheable loads to memory mapped using page tables located in write combining memory may return incorrect data. Intel has not been able to reproduce this erratum with commercially available software.

Workaround: Do not place page directories and/or page tables in WC memory.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V26. Buffer On Resistance May Exceed Specification

Problem: The datasheet specifies the resistance range for R_{ON} (Buffer On Resistance) for the AGTL+ and Asynchronous GTL+ buffers as 5 to 11 ohms. Due to this erratum, R_{ON} may be as high as 13.11 ohms.

Implication: The R_{ON} value affects the voltage level of the signals when the buffer is driving the signal low. A higher R_{ON} may adversely affect the system's ability to meet specifications such as V_{IL}. As the system design also affects margin to specification, designs may or may not have sufficient margin to function properly with an increased R_{ON}. System designers should evaluate whether a particular system is affected by this erratum. Designs that follow the recommendations in the *Intel® Pentium® 4 Processor and Intel® 850 Chipset Platform Design Guide* are not expected to be affected.

Workaround: No workaround is necessary for systems with margin sufficient to accept a higher R_{ON}.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V27. Processor Issues Inconsistent Transaction Size Attributes for Locked Operation

Problem: When the processor is in the Page Address Extension (PAE) mode and detects the need to set the Access and/or Dirty bits in the page directory or page table entries, the processor sends an 8 byte load lock onto the system bus. A subsequent 8 byte store unlock is expected, but instead a 4 byte store unlock occurs. Correct data is provided since only the lower bytes change, however external logic monitoring the data transfer may be expecting an 8 byte store unlock.

Implication: No known commercially available chipsets are affected by this erratum.

Workaround: None identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V28. Multiple Accesses to the Same S-State L2 Cache Line and ECC Error Combination May Result in Loss of Cache Coherency

Problem: When a Read for Ownership (RFO) cycle has a 64 bit address match with an outstanding read hit on a line in the L2 cache which is in the S-state AND that line contains an ECC error, the processor should recycle the RFO until the ECC error is handled. Due to this erratum, the processor does not recycle the RFO and attempts to service both the RFO and the read hit at the same time.

Implication: When this erratum occurs, cache may become incoherent.

Workaround: None identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V29. Processor May Hang When Resuming From Deep Sleep State

Problem: When resuming from the Deep Sleep state the address strobe signals (ADSTB[1:0]#) may become out of phase with respect to the system bus clock (BCLK).

Implication: When this erratum occurs, the processor will hang.

Workaround: The system BIOS should prevent the processor from going to the Deep Sleep state.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V30. When the Processor is in the System Management Mode (SMM), Debug Registers May be Fully Writeable

Problem: When in System Management Mode (SMM), the processor executes code and stores data in the SMRAM space. When the processor is in this mode and writes are made to DR6 and DR7, the processor should block writes to the reserved bit locations. Due to this erratum, the processor may not block these writes. This may result in invalid data in the reserved bit locations.

Implication: Reserved bit locations within DR6 and DR7 may become invalid.

Workaround: Software may perform a read/modify/write when writing to DR6 and DR7 to ensure that the values in the reserved bits are maintained.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V31. Associated Counting Logic Must be Configured When Using Event Selection Control (ESCR) MSR

Problem: ESCR MSRs allow software to select specific events to be counted, with each ESCR usually associated with a pair of performance counters. ESCRs may also be used to qualify the detection of at-retirement events that support precise-event-based sampling (PEBS). A number of performance metrics that support PEBS require a 2nd ESCR to tag uops for the qualification of at-retirement events. (The first ESCR is required to program the at-retirement event.) Counting is enabled via counter configuration control registers (CCCR) while the event count is read from one of the associated counters. When counting logic is configured for the subset of at-retirement events that require a 2nd ESCR to tag uops, at least one of the CCCRs in the same group of the 2nd ESCR must be enabled.

Implication: If no CCCR/counter is enabled in a given group, the ESCR in that group that is programmed for tagging uops will have no effect. Hence a subset of performance metrics that require a 2nd ESCR for tagging uops may result in 0 count.

Workaround: Ensure that at least one CCCR/counter in the same group as the tagging ESCR is enabled for those performance metrics that require two ESCRs and tagging uops for at-retirement counting.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V32. IA32_MC0_ADDR and IA32_MC0_MISC Registers Will Contain Invalid or Stale Data Following a Data, Address, or Response Parity Error

Problem: If the processor experiences a data, address, or response parity error, the ADDRV and MISCV bits of the IA32_MC0_STATUS register are set, but the IA32_MC0_ADDR and IA32_MC0_MISC registers are not loaded with data regarding the error.

Implication: When this erratum occurs, the IA32_MC0_ADDR and IA32_MC0_MISC registers will contain invalid or stale data.

Workaround: Ignore any information in the IA32_MC0_ADDR and IA32_MC0_MISC registers after a data, address or response parity error.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V33. CR2 May Be Incorrect or an Incorrect Page Fault Error Code May Be Pushed onto Stack After Execution of an LSS Instruction

Problem: Under certain timing conditions, the internal load of the selector portion of the LSS instruction may complete with potentially incorrect speculative data before the load of the offset portion of the address completes. The incorrect data is corrected before the completion of the LSS instruction but the value of CR2 and the error code pushed on the stack are reflective of the speculative state. Intel has not observed this erratum with commercially available software.

Implication: When this erratum occurs, the contents of CR2 may be off by two, or an incorrect page fault error code may be pushed onto the stack.

Workaround: It is possible for the BIOS to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V34. System May Hang if a Fatal Cache Error Causes Bus Write Line (BWL) Transaction to Occur to the Same Cache Line Address as an Outstanding Bus Read Line (BRL) or Bus Read-Invalidate Line (BRIL)

Problem: A processor internal cache fatal data ECC error may cause the processor to issue a BWL transaction to the same cache line address as an outstanding BRL or BRIL. As it is not typical behavior for a single processor to have a BWL and a BRL/BRIL concurrently outstanding to the same address, this may represent an unexpected scenario to system logic within the chipset.

Implication: The processor may not be able to fully execute the machine check handler in response to the fatal cache error if system logic does not ensure forward progress on the system bus under this scenario.

Workaround: System logic should ensure completion of the outstanding transactions. Note that during recovery from a fatal data ECC error, memory image coherency of the BWL with respect to BRL/BRIL transactions is not important. Forward progress is the primary requirement.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section

V35. Processor Does not Flag #GP on Non-zero Write to Certain MSRs

Problem: When a non-zero write occurs to the upper 32 bits of IA32_CR_SYSENTER_EIP or IA32_CR_SYSENTER_ESP, the processor should indicate a general protection fault by flagging #GP. Due to this erratum, the processor does not flag #GP.

Implication: The processor unexpectedly does not flag #GP on a non-zero write to the upper 32 bits of IA32_CR_SYSENTER_EIP or IA32_CR_SYSENTER_ESP. No known commercially available operating system has been identified to be affected by this erratum.

Workaround: None identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V36. L2 Cache May Contain Stale Data in the Exclusive State

Problem: If a cacheline (A) is in Modified (M) state in the write-combining (WC) buffers and in the Invalid (I) state in the L2 cache and its adjacent sector (B) is in the Invalid (I) state and the following scenario occurs:

1. A read to B misses in the L2 cache and allocates cacheline B and its associated second-sector pre-fetch into an almost full bus queue,
2. A Bus Read Line (BRL) to cacheline B completes with HIT# and fills data in Shared (S) state,
3. The bus queue full condition causes the prefetch to cacheline A to be cancelled, cacheline A will remain M in the WC buffers and I in the L2 while cacheline B will be in the S state.

Then, if the further conditions occur:

1. Cacheline A is evicted from the WC Buffers to the bus queue which is still almost full,
2. A hardware prefetch Read for Ownership (RFO) to cacheline B, hits the S state in the L2 and observes cacheline A in the I state, allocates both cachelines,
3. An RFO to cacheline A completes before the WC Buffers write modified data back, filling the L2 with stale data,
4. The writeback from the WC Buffers completes leaving stale data, for cacheline A, in the Exclusive (E) state in the L2 cache.

Implication: Stale data may be consumed leading to unpredictable program execution. Intel has not been able to reproduce this erratum with commercial software.

Workaround: It is possible for BIOS to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

V37. Simultaneous Assertion of A20M# and INIT# may Result in Incorrect Data Fetch

Problem: If A20M# and INIT# are simultaneously asserted by software, followed by a data access to the 0xFFFFFXXX memory region, with A20M# still asserted, incorrect data will be accessed. With A20M# asserted, an access to 0xFFFFFXXX should result in a load from physical address 0xFFEFFXXX. However, in the case of A20M# and INIT# being asserted together, the data load will actually be from the physical address 0xFFFFFXXX. Code accesses are not effected by this erratum.

Implication: Processor may fetch incorrect data, resulting in BIOS failure.

Workaround: Deasserting and reasserting A20M# prior to the data access will workaround this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

.

V38. Glitches on Address or Data Strobe Signals May Cause System Shutdown

Problem: When a Machine Check Exception is generated due to a glitch on the address or data strobe signals, the exception may be reported repeatedly, resulting in system shutdown

Implication: If a glitch occurs on the address or data strobe signals, an operating system shutdown will occur if Machine Check Exceptions (MCE) are enabled. IERR# assertion and shutdown will occur if MCE is disabled.

Workaround: Correct design and implementation of the processor system bus will remove the possibility of this failure.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the following documents:

- *Intel® Celeron® Processor in the 478-Pin Package Datasheet*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3(Order Numbers 245470, 245471, and 245472, respectively)*

All Documentation Changes will be incorporated into a future version of the appropriate Intel® Celeron® processor documentation.

V1. SSE and SSE2 Instructions Opcodes

The note at the end of section 2.2 in the *Intel Architecture Software Developer's Manual*, *Vol 2*: *Instruction Set Reference* states:

> NOTE:
> Some of the SSE and SSE2 instructions have three-byte opcodes. For these three-byte opcodes, the third opcode byte may be F2H, F3H, or 66H. For example, the SSE2 instruction CVTDQ2PD has the three-byte opcode F3 OF E6. The third opcode byte of these three-byte opcodes should not be thought of as a prefix, even though it has the same encoding as the operand size prefix (66H) or one of the repeat prefixes (F2H and F3H). As described above, using the operand size and repeat prefixes with SSE and SSE2 instructions is reserved.

It should state:

> NOTE:
> Some of the SSE and SSE2 instructions have three-byte opcodes. For these three-byte opcodes, the third opcode byte may be F2H, F3H, or 66H. For example, the SSE2 instruction CVTDQ2PD has the three-byte opcode F3 OF E6. The third opcode byte of these three-byte opcodes should not be thought of as a prefix, even though it has the same encoding as the operand size prefix (66H) or one of the repeat prefixes (F2H and F3H). As described above, using the operand size and repeat prefixes with SSE and SSE2 instructions is reserved. It should also be noted that execution of SSE2 instructions on an Intel processor that does not support SSE2 (CPUID Feature flag register EDX bit 26 is clear) will result in unpredictable code execution.

V2. Executing the SSE2 Variant on a Non-SSE2 Capable Processor

In *Intel Architecture Software Developer's Manual, Vol 2*: *Instruction Set Reference* the section for each of the following instructions states that executing the instruction in real or protected mode on a processor for which the SSE2 feature flag returned by CPUID is 0 (SSE2 not supported by the processor) will result in the generation of an undefined opcode exception (#UD). This is incorrect. The SSE2 form of these instructions is defined by opcodes for which the leading opcode byte maps into an operand size prefix. Executing the SSE2 variant of these instructions on a non-SSE2 capable processor will result in an SSE like operation and not a #UD. Refer to section 2.2 of the *Intel Architecture Software Developer's Manual, Vol 2* for more detail.

Instructions:
MOVD xmm, r32; MOVD r32, xmm; MOVDQA; MOVDQU; MOVQ xmm, m64; PACKSSWB/DW; PACKUSWB; PADDB/W/D; PADDSB/W; PADDUSB/W; PAND; PANDN; PCMPEQB/W/D; PCMPGTB/W/D; PMADDWD; PMULHW; PMULLW; POR; PSLLW/D/Q; PSRAW/D; PSRLW/D/Q; PSUBB/W/D; PSUBSB/W; PSUBUSB/W; PUNPCKHBW/WD/DQ; PUNPCKLBW/WD/DQ; PXOR.

V3. Direction Flag (DF) Mistakenly Denoted as a System Flag

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 3.4.3 "EFLAGS Register", in Figure 3-7. EFLAGS Register currently states:

X Direction Flag(DF)

It should state:

C Direction Flag(DF)

V4. Fopcode Compatibility Mode

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 8.1.8.1 "FOPCODE COMPATIBILITY MODE" currently states:

> "When the FOP code compatibility mode is enabled, the IA32 architecture guarantees that if an unmasked x87 FPU floating-point exception is generated, the opcode of the last non-control instruction executed prior to the generation of the exception will be stored in the x87 FPU opcode register, and that value can be read by a subsequent FSAVE of FXSAVE instruction. When the fop compatibility mode is disabled (default), the value stored in the x87 FPU opcode register is undefined (reserved)."

It should state:

> "If FOP code compatibility mode is enabled, the FOP is defined as it has always been in previous IA32 implementations (always defined as the FOP of the last non-transparent FP instruction executed before a FSAVE/FSTENV/FXSAVE).
>
> If FOP code compatibility mode is disabled (default), FOP is only valid if the last non-transparent FP instruction executed before a FSAVE/FSTENV/FXSAVE had an unmasked exception."

V5. FCOS, FPTAN, FSIN, and FSINCOS Trigonometric Domain NOT correct.

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Section 3.2 "INSTRUCTION REFERENCE" FCOS, FPTAN, FSIN, and FSINCOS trigonometric domain for C2 is incorrect. Under the FPU Flags affected, C2 currently states:

C2 Set to 1 if source operand is outside the range -2^{63} to $+2^{63}$; otherwise, cleared to 0.

It should state:

C2 Set to 1 if outside range -2^{63} <source operand <$+2^{63}$; otherwise, set to 0.

V6. Incorrect Description of stack

The *IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture* Chapter 6, Section 6.2 paragraph 2, labeled "STACK" currently states:

> The next available memory location on the stack is called the top of stack. At any given time, the stack pointer (contained in the ESP register) gives the address (that is the offset from the base of the SS segment) of the top of the stack.

This paragraph is incorrect and will be removed from the section listed above.

V7. EFLAGS Register Correction

The *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture* Section 3.7.2, Figure 3.7. "EFLAGS Register" currently states:

Bit 11 "OF" as "X"

It should state:

Bit 11 "OF" as "S"

V8. PSE-36 Paging Mechanism

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* Chapter 3, Section 3.9, third paragraph currently states:

As is shown in Table 3-3, the following flags must be set or cleared to enable the PSE-36 paging mechanism:

- PSE-36 CPUID feature flag-When set, it indicates the availability of the PSE-36 paging mechanism on the IA-32 processor on which the CPUID instruction is executed.
- PG flag (bit 31) in register CR0-Set to 1 to enable paging.
- PSE flag (bit 4) in control register CR4 - Set to 1 to enable the page size extension for 4-Mbyte pages.
- PAE flag (bit 5) in control register CR4-Clear to 0 to disable the PAE paging mechanism.

It should state:

As is shown in Table 3-3, the following flags must be set or cleared to enable the PSE-36 paging mechanism:

- PSE-36 CPUID feature flag-When set, it indicates the availability of the PSE-36 paging mechanism on the IA-32 processor on which the CPUID instruction is executed.
- PG flag (bit 31) in register CR0-Set to 1 to enable paging.
- PAE flag (bit 5) in control register CR4-Clear to 0 to disable the PAE paging mechanism.
- PSE flag (bit 4) in control register CR4 and the PS flag in PDE- Set to 1 to enable the page size extension for 4-Mbyte pages.
- Or the PSE flag (bit 4) in control register CR4- Set to 1 and the PS flag (bit 7) in PDE- Set to 0 to enable 4-KByte pages with 32-bit addressing (below 4GBytes).

V9. 0x33 Opcode

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Appendix A, Table A-2, the opcode corresponding to 0x33 currently states:

Gb, Ev

It should state:

Gv, Ev

Also, Page 3-791, XOR-Logical Exclusive OR, the two entries for opcode 33 currently state:

Opcode	Instruction	Description
33 /r	XOR r16,r/m16	r8 XOR r/m8
33 /r	XOR r32,r/m32	r8 XOR r/m8

It should state:

Opcode	Instruction	Description
33 /r	r16 XOR r/m16	r8 XOR r/m8
33 /r	r32 XOR r/m32	r8 XOR r/m8

V10. Incorrect Information for SLDT

In the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference,* the opcode/Instruction/Description table for SLDT currently states "SLDT *r/m32* Store segment selector from LDTR in low-order 16 bits of *r/m32*" but should instead list "SLDT *r32* Store segment selector from LDTR in low-order 16 bits of *r32*."

V11. LGDT/LIDT Instruction Information Correction

In the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference,* the sentence in the LGDT/LIDT instruction section currently states:

"See 'SFENCE -- Store Fence' in this chapter for information on storing the contents of the GDTR and IDTR."

It should state:

"See 'SGDT/SIDT' in this chapter for information on storing the contents of the GDTR and IDTR."

V12. Errors in Instruction Set Reference

The following changes will be made to the *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference:*

1. *Page 3-586 "PMULUDQ—Multiply Packed Unsigned Doubleword Integers"* currently states:

   66 OF F4 /r PMULUDQ xmm1, xmm2/m128

   It should state:

   66 0F F4 /r PMULUDQ xmm1, xmm2/m128

2. *Page A-9, Table A-3, Two-byte Opcode Map:08H-7FH (First Byte is 0FH), e*ntry 2B currently states:

   MOVNTPS
   Wps, Vps
   MOVNTPS (66)
   Wpd, Vpd

   It should state:

   MOVNTPS
   Wps, Vps
   MOVNTPD (66)
   Wpd, Vpd

3. *Page A-9, Table A-3, Two-byte Opcode Map:08H-7FH (First Byte is 0FH).*
   Entry 3C currently states:

   Blank (empty space)

   It should state:

   MOVNTI

4. *Page A-10, Table A-3, Two-byte Opcode Map:80H-7FH (First Byte is 0FH).*
   Entry D7 currently states:

   PMOVMSKB
   Gd, Pq
   PMOVMKSB (66)
   Gd, Vdq

   It should state:

   PMOVMSKB

Gd, Pq
PMOVMSKB (66)
Gd, Vdq

5. *Page A-10, Table A-3, Two-byte Opcode Map:80H-7FH (First Byte is 0FH).*
Entry F7 currently states:

MASKMOVQ
Ppi, Qpi
MASKMOVQU (66)
Vdq, Wdq

It should state:

MASKMOVQ
Ppi, Qpi
MASKMOVDQU (66)
Vdq, Wdq

6. *Page A-11, Table A-3*, *Two-byte Opcode Map:88H-7FH (First Byte is 0FH).*
The title table currently states:

Table A-3. Two-byte Opcode Map:88H-7FH (First Byte is FFH)

It should state:

Table A-3. Two-byte Opcode Map:88H-7FH (First Byte is 0FH)

7. *Page A-11, Table A-3*, *Two-byte Opcode Map:88H-7FH (First Byte is 0FH).*
Entry FB currently states:

PSUBD
Pq, Qq
PSUBD (66)
Vdq, Wdq

It should state:

PSUBQ
Pq, Qq
PSUBQ (66)
Vdq, Wdq

8. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
Entry PMADD currently states:

PMADD – Packed Multiply add

It should state:

PMADDWD – Packed Multiply add

9. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*

Entry PMULH currently states:

PMULH – Packed multiplication

It should state:

PMULHW – Packed multiplication, store high word

10. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
Add instruction PMULHUW :

PMULHUW – Packed multiplication, store high word (unsigned)

mmxreg2 to mmxreg1	0000 1111: 1110 0100: 11 mmxreg1 mmxreg2
memory to mmxreg	0000 1111: 1110 0100: mod mmxreg r/m

11. *Page B-21, Table B-12, MMX Instruction Formats and Encodings (Contd.).*
Entry PMULL currently states:

PMULL – Packed multiplication

It should state:

PMULLW – Packed multiplication, store low word

12. *Page B-40, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Entry PMADD currently states:

PMADD – Packed multiply add

It should state:

PMADDWD – Packed multiply add

13. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Entry PMULH currently states:

PMULH – Packed multiplication

It should state:

PMULHW – Packed multiplication, store high word

14. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Add instruction PMULHUW:

PMULHUW – Packed multiplication, store high word (unsigned)

xmmreg2 to xmmreg1	0110 0110 : 0000 1111 : 11110 0100 : 11 xmmreg1 xmmreg2

memory to xmmreg 0110 0110 : 0000 1111 : 1110 0100 : mod xmmreg r/m

15. *Page B-41, Table B-19, Formats and Encodings of the SSE2 SIMD Integer Instruction.*
Entry PMULL currently states:

PMULL – Packed multiplication

It should state:

PMULLW – Packed multiplication, store low word

V13. RSM Instruction Set Summary

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture* Section 5.8 "INSTRUCTION SET SUMMARY" currently states:

RSM Return from system management mode (SSM)

It should state:

RSM Return from system management mode (SMM)

V14. Correct MOVAPS and MOVAPD Operand Section

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Section 3.2 "INSTRUCTION REFERENCE" MOVAPS and MOVAPD operation section currently states:

Operation

DEST ← SRC;

It should state:

Operation

DEST ← SRC;

- #GP if SRC or DEST unaligned memory operand *;

V15. DAA—Decimal Adjust AL after Addition

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* page 3-173 currently states:

Operation

```
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL + 6;
    CF ←CF OR CarryFromLastAddition; (* CF OR carry from AL ←AL + 6 *)
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((AL AND F0H) > 90H) or CF = 1)
  THEN
    AL ←AL + 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

It should state:

Operation

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL + 6;
    CF ←old_CF or (Carry from AL ←AL + 6);
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((old_AL > 99H) or (old_CF = 1)
  THEN
    AL ←AL + 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

V16. DAS—Decimal Adjust AL after Subtraction

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference*, on page 3-175 currently states:

Operation

```
IF (AL AND 0FH) > 9 OR AF = 1
  THEN
    AL ←AL - 6;
    CF ←CF OR CarryFromLastAddition; (* CF OR carry from AL ←AL - 6 *)
    AF ←1;
  ELSE AF ←0;
FI;
IF ((AL > 9FH) or CF = 1)
  THEN
    AL ←AL - 60H;
    CF ←1;
  ELSE CF ←0;
FI;
```

It should state:

Operation

```
old_AL ← AL;
old_CF ← CF;
CF ← 0;
IF (((AL AND 0FH) > 9) or AF = 1)
  THEN
    AL ←AL - 6;
    CF ←old_CF or (Borrow from AL ←AL - 6);
    AF ←1;
  ELSE
    AF ←0;
FI;
IF ((old_AL > 99H) OR (old_CF = 1))
  THEN
    AL ←AL - 60H;
    CF ←1;
  ELSE
    CF ←0;
FI;
```

V17. Omission of Dependency Between BTM and LBR

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* Chapter *15*, Section 5.3, on page 15-15 currently states:

> **15.5.3. Monitoring Branches, Exceptions, and Interrupts (Pentium 4 and Intel Xeon Processors)**
> When the LBR flag in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions)
> in the LBR stack MSRs.
>
> When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the LBR stack MSRs. The branch records for the last four taken branches, interrupts, and/or exceptions are thus retained for anal-ysis by the debugger program.
>
> The debugger can use the linear addresses in the LBR stack to reset breakpoints in the break-point-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source.
>
> Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch recording.

It should state:

> **15.5.3. Monitoring Branches, Exceptions, and Interrupts (Pentium 4 and Intel Xeon Processors)**
> When the LBR flag in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions)
> in the LBR stack MSRs.
>
> When the processsor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs. The branch record for the last four taken branches, interrupts and/or exceptions are retained for analysis.
>
> A debugger can use the linear addresses in the LBR stack to reset breakpoints in the break-point address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.
>
> If the LBR flag is cleared and TR flag in the IA32_DEBUGCTLTR MSR remains set, the processor will continue to update LBR stack MSRs. This is because BTM information must be generated from entries in the LBR stack (see 14.5.5). A #DB does not automatically clear the TR flag.

V18. I/O Permissions Bitmap Base Addy > 0xDFFF Does not Cause #GP(0) Fault

The *Intel Architecture Software Developer's Manual, Vol 1: Basic Architecture,* page 12-6, section 12.5.2, last paragraph currently states:

> If the I/O bit map base address is greater than or equal to the TSS segment limit, there is no I/O permission map, and all I/O instructions generate exceptions when the CPL is greater than the current IOPL. The I/O bit map base address must be less than or equal to DFFFH.

It should state:

> If the I/O bit map base address is greater than or equal to the TSS segment limit, there is no I/O permission map, and all I/O instructions generate exceptions when the CPL is greater than the current IOPL.

V19. Wrong Field Width for MINSS and MAXSS

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference Section 3.2 Instruction Reference* under "MAXSS—Return Maximum Scalar Single-Precision Floating-Point Value" page 3-415 currently states:

> DEST[63-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

It should state:

> DEST[31-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

The *Intel Architecture Software Developer's Manual, Vol 2 Section 3.2 Instruction Reference* under title "MINSS—Return Minimum Scalar Single-Precision floating-Point Value" page 3-428 currently states:

> DEST[63-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

It should state:

> DEST[31-0] .IF ((DEST[31-0] = 0.0) AND (SRC[31-0] = 0.0)) THEN SRC[31-0]

V20. Figure 15-12 PEBS Record Format

The *Intel Architecture Software Developer's Manual, Vol 3*: System Programming Guide *Section 15.9.6 " Programming the Performance Counters for Non-Retirement Events"* page 15 - 37, Figure 15-12, first row currently states:

It should state:

V21. I/O Permission Bit Map

The *Intel Architecture Software Developer's Manual, Vol 1:* Basic Architecture Chapter 12, section 12.5.2 on Figure 12-2 (I/O Permission Bit Map) currently states:

Last byte of bit map must be followed by a byte with all bits.

It should state:

Last byte of bit map must be followed by a byte with all bits set.

Also, in the lower left hand Conner of Figure 12-2 (I/O Permission Bit Map) *currently states:*

Last I/O base map must be

It should state:

Last I/O base map must be less than or equal to DFFFH

V22. Cache Description

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Table 3-10, the "sectored, 64 byte line size" description is used for the following descriptors: 0x22, 0x23, 0x79, 0x7a, 0x7b, 0x7c. This description will change to "dual-sectored line, 64 byte sector size" for clarity.

V23. Instruction Formats and Encoding

The *Intel Architecture Software Developer's Manual, Vol 2: Instruction Set Reference* Page B-8, CMOVcc memory to register should be encoded as "0000 1111 : 0100 tttn : mod reg r/m". Page B-8, CMP immediate with memory should be encoded as "1000 00sw : mod 111 r/m : immediate data". Page B-12 POP "segment register CS, DS, ES" should be encoded as "segment register DS, ES".

V24. Machine-Check Initialization

The *Intel Architecture Software Developer's Manual, Vol 3: System Programming Guide* section 14.5 currently states*:*

14.5 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception on the processor, then enables the machine-check exception and the error-reporting register banks. The pseudocode assumes that the machine-check exception (#MC) handler has been installed on the system. This initialization procedure is compatible with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Following power up or power cycling, the IA32_MC*i*_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all 0s by software, as shown in the initialization pseudocode in Example .

Machine-Check Initialization Pseudocode

EXECUTE the CPUID instruction;

```
READ bits 7 (MCE) and 14 (MCA) of the EDX register;
IF CPU supports MCE
    THEN
        IF CPU supports MCA
            THEN
                IF IA32_ MCG_CAP.MCG_CTL_P = 1
                (* IA32_MCG_CTL register is present *)
                    IA32_MCG_CTL  FFFFFFFFFFFFFFFFH;
                    (* enables all MCA features *)
                FI;
                COUNT <-- IA32_MCG_CAP.Count;
                MAX_BANK_NUMBER <-- COUNT~ 1;
                (* determine number of error-reporting banks supported *)
                IF (P6 Family Processor)
                    THEN
                        FOR error-reporting banks (1 through MAX_BANK_NUMBER) DO
                            IA32_MCi_CTL <-- FFFFFFFFFFFFFFFFH;
                            (* enables logging of all errors except for MC0_CTL register *)
                        OD
                    ELSE (* Pentium 4 and Intel Xeon Processors *)
                        FOR error-reporting banks (0 through MAX_BANK_NUMBER) DO
                            IA32_MCi_CTL <-- FFFFFFFFFFFFFFFFH;
                            (* enables logging of all errors including MC0_CTL register *)
                        OD
                FI;
                FOR error-reporting banks (0 through MAX_BANK_NUMBER) DO
                    IA32_MCi_STATUS <-- 0000000000000000H; (* clears all errors *)
                OD
        FI;
    Set the MCE flag (bit 6) in CR4 register to enable machine-check exceptions;
FI;
```

It should state:

14.5 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception on the processor, then enables the machine-check exception and the error-reporting register banks. The pseudocode shown is compatible with the Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Following power up or power cycling, the IA32_MC*i*_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all 0s by software, as shown in the initialization pseudocode in Example . In addition, when using P6 family processors, the software must set MCi_STATUS registers to 0 when doing a soft-reset.

Machine-Check Initialization Pseudocode

```
Check CPUID Feature Flags for MCE and MCA support
IF CPU supports MCE
THEN
    IF CPU supports MCA
    THEN
        IF (IA32_MCG_CAP.MCG_CTL_P = 1)
        (* IA32_MCG_CTL register is present *)
        THEN
            IA32_MCG_CTL <-- FFFFFFFFFFFFFFFFH;
            (* enables all MCA features *)
        FI

        (* Determine number of error-reporting banks supported *)
        COUNT<-- IA32_MCG_CAP.Count;
        MAX_BANK_NUMBER <-- COUNT - 1;

        IF (Processor Family is 6H)
        THEN
            (* Enable logging of all errors except for MC0_CTL register *)
            FOR error-reporting banks (1 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL <-- 0FFFFFFFFFFFFFFFFH;
            OD

            (* Clear all errors *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_STATUS <-- 0;
            OD

        ELSE IF (Processor Family is 0FH) (*any Processor Extended Family *)
        THEN
            (* Enable logging of all errors including MC0_CTL register *)
            FOR error-reporting banks (0 through MAX_BANK_NUMBER)
            DO
                IA32_MCi_CTL <-- 0FFFFFFFFFFFFFFFFH;
            OD

            (* BIOS clears all errors only on power-on reset *)
```

```
            IF (BIOS detects Power-on reset)
            THEN
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    IA32_MCi_STATUS <-- 0;
                OD
            ELSE
                FOR error-reporting banks (0 through MAX_BANK_NUMBER)
                DO
                    (Optional for BIOS and OS) Log valid errors
                    (OS only) IA32_MCi_STATUS <-- 0;
                OD

            FI
        FI
FI

Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions
FI
```

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the following documents:

- *Intel® Celeron® Processor in the 478-Pin Package Datasheet*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3 (Order Numbers 245470, 245471, and 245472, respectively)*

All Specification Clarifications will be incorporated into a future version of the appropriate Intel® Celeron® processor documentation.

There are no specification clarifications to report.

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the following documents:

- *Intel® Celeron® Processor in the 478-Pin Package Datasheet*
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3 (Order Numbers 245470, 245471, and 245472, respectively)*

All Specification Changes will be incorporated into a future version of the appropriate Intel® Celeron® processor documentation.

There are no specification changes to report.